Università di Roma TorVergata
Laurea Magistrale in Informatica

Corso: Modelli e Qualità del Software

Giulia Pascale, 0316810
scriba.fox@gmail.com
(oppure:
giulia.pascale@students.uniroma2.eu)

9 CFU

Nome della prova: Prit2 MQS
Data della Prova: 05/06/2024

Domande

1) Stimare il tasso iniziale □0 di failure per difetto, simulando un **pre-run** con lifetime Ti esponenziali di tasso □**i = 0.00628** failure/ora, identico per tutti gli ***i***, e di durata □□sufficiente a produrre **n=125 failure**. Si assuma l'esistenza di **N0= 48** iniziali difetti latenti (dire come si pensa di aver stimato N0). Per il generatore pseudocasuale della sequenza di base < Xi > da cui ottenere la sequenza dei < Ti > si usino i dati indicati in NOTA.

2) Simulare poi un **run** (*) con Ti esponenziali di tasso □i (failure/ora) calcolato, ad ogni i, secondo J&M e di durata sufficiente a ottenere 50 failure, e rilevare la sequenza (T1, …, T50) dei tempi di lifetime.

3) Sulla base di detta sequenza, usando maximum likelihood e modello J&M stimare i parametri N e □ necessari alla stima di affidabilità, adottando il procedimento Newton-Raphson (*codice Math* accluso) per la ricerca delle radici, e usando come iniziali i valori No , □□□ di cui sopra, e numero di iterazioni come indicato in NOTA.

4) Dire cosa rappresentano i parametri N e □ stimati (indicandone le metriche) e come si perviene ad esprimere le funzioni **f1** e **f2** del codice Math richieste dal procedimento Newton-Raphson.

5) Con i valori di N e □ sopra stimati, esprimere e calcolare con J&M il numero *i* di failures che si dovranno sperimentare prima di eliminare il 90% dei fault, e il relativo tempo medio E(ti) necessario (indicandone la metrica).

6) Dare con J&M l'espressione della affidabilita' Ri(t) del prodotto dopo l'eliminazione dell'i-esimo fault e con essa calcolare la probabilità che la successiva failure non si verifichi prima di 3 ore, e quella che non si verifichi prima di 9 ore.

(*) La sequenza < Xi > per la simulazione del **run** sia il seguito immediato della sequenza del **pre-run**.

NOTA

- Dati per la sequenza pseudocasuale < Xi > di base:

Generatore moltiplicativo con a = 1.220.703.125 m=2.147.483.648, seme X0 = da assicurare periodo max

- Numero di iterazioni nel NewRaph = fino a convergenza con distanza tra radici successive inferiore a 10-6.

Risposte

Risposta 1

Partiamo col dire come si pensa di aver **stimato il valore N_0**, cioè il valore di difetti iniziali latenti. Quindi, prima di stimare e simulare i dati richiesti, iniziamo con l'introdurre le metriche del prodotto software, utilizzate per valutare la complessità di un software. Queste metriche si suddividono principalmente in due categorie: le **metriche orientate ai processi (process oriented), che valutano la complessità di progetti dettagliati che vedano il software come un insieme di moduli**, e le **metriche orientate agli oggetti (object oriented), che si focalizzano su progetti dettagliati che vedano il prodotto come un insieme di oggetti.**

Per quanto riguarda le process oriented si parla di D-structuredness, Depth of nesting e Cyclomatic complexity. Per quanto riguarda le object oriented si parla di PD, CBO, LOCM, WMC, Depth, RFC, OOFI, CDC, NOC. Tutte queste metriche possono essere usate per la ***stima statica del numero iniziale*** **N_0** ***di difetti introdotti nel modulo in fase di codifica***. Infatti, minore è la complessità, minore saranno i difetti introdotti **N_0**.

Quando si procede alla codifica di due progetti dettagliati, PD1 e PD2, relativi allo stesso modulo, il codice con un valore più basso di **N_0** sarà presunto (con una certa attendibilità) come quello che presenterà migliori caratteristiche:

- Struttura più ordinata (migliore d-structuredness d(F))
- Meno nidificazioni (minore depth of nesting a(F))
- Complessità ciclomatica più bassa (minore cyclomatic complexity cc(F))

I **modelli statici di affidabilità sono strumenti che**, mediante l'analisi regressiva di dati provenienti da progetti già esistenti, permettono di **stabilire una relazione tra le misure di complessità del software e il numero iniziale di difetti**. Alcuni metodi statici visti sono: Akiyama, Brooks, Halstead, Lipow, Gaffney, Compton-Withrow, Rodriguez-Harrison-Satpathy-Dolado, Sherry-Malviya-Tripathi.

Infine diciamo che per verificare l'attendibilità delle stime, vengono confrontati:

- Il numero reale (y) di difetti presenti nel prodotto (introdotto artificialmente)
- Il numero stimato ($\hat{y}$), indicato precedentemente come **N_0**.

Un modo per effettuare questo confronto è basata sul calcolo di Errore relativo medio(RE) e Correlazione(CR).

1.1) Il testing Statico è basato sul **profilo operativo**. Vengono simulate una serie di esecuzioni del modulo ottenendo una o più serie di tempi di inter-failure. Questo avviene in due fasi: ***pre-run*** e ***run. Nella f*****ase** ***pre-run c'è la*** **generazione dei tempi di interfailure**.

In particolare la fase di pre-run ha una durata di esecuzione sufficientemente lunga τ per generare un numero n (es n=125) di tempi di lifetime $T_1, T_2, \ldots, T_{125}$ e si occupa del calcolo di diversi valori come i valori iniziali (**N_0**, φ_0) da usare nella fase di run.

Nel caso in esame assumiamo di avere $N_0 = 48$. Eseguiamo un pre-run generando n = 125 tempi di lifetime T_i di tasso $\lambda_i = 0.000628$ (failure/ora), uguale per tutti gli i.

Vengono quindi generati n tempi di interfailure esponenziali T_i.

Si calcolano quindi i valori:

- tempo totale τ (ore) di prerun (somma di tutti i tempi di lifetime): $\tau = \sum_{i=0}^{n-1} T_i$
- tasso complessivo di failure (failure/ora) $\Phi_0 = n / \tau$
- tasso iniziale di failure per difetto (failure/ora) $\varphi_0 = \Phi_0 / N_0$

Di seguito il codice Java con la stampa dei tempi di interfailure e dei valori calcolati.

In breve, quello che è stato fatto nel codice, è stato costruire la classe PreRun che contenesse gli attributi necessari alla simulazione della fase di prerun, con la generazione dei tempi di interfailure.

Successivamente si hanno due funzioni:

- *prerun():* si occupa di generare i tempi di interfailure utilizzando il generatore random di numeri esponenziali di tasso lambda.
- *calcolophi()*: si occupa di calcolare il valore di φ_0.

Il main della classe si occupa di eseguire il tutto e di stamparlo a schermo, in modo tale da visualizzare i Lifetimes e il valore di φ_0.

```java
package Prit2_MQS_Pascale;

import java.util.Arrays;

public class PreRun {

    private static final double lambda_i = 0.00628;
    private static final int N_0 = 48;
    private static final int n_Failure = 125;
    private static final long seme = 57;
    private double totalTime = 0.0;
    GeneratoreCasuale RandomGenerator;

    public double[] prerun() {

        int failureCount = 0;
        double[] lifetimes = new double[n_Failure]; //array per T

        RandomGenerator = new GeneratoreCasuale(1/lambda_i, seme);

        while (failureCount < n_Failure) {
            double lifetime = RandomGenerator.getNumberExp();

            lifetimes[failureCount] = lifetime;
            totalTime += lifetime;
            failureCount++;
```

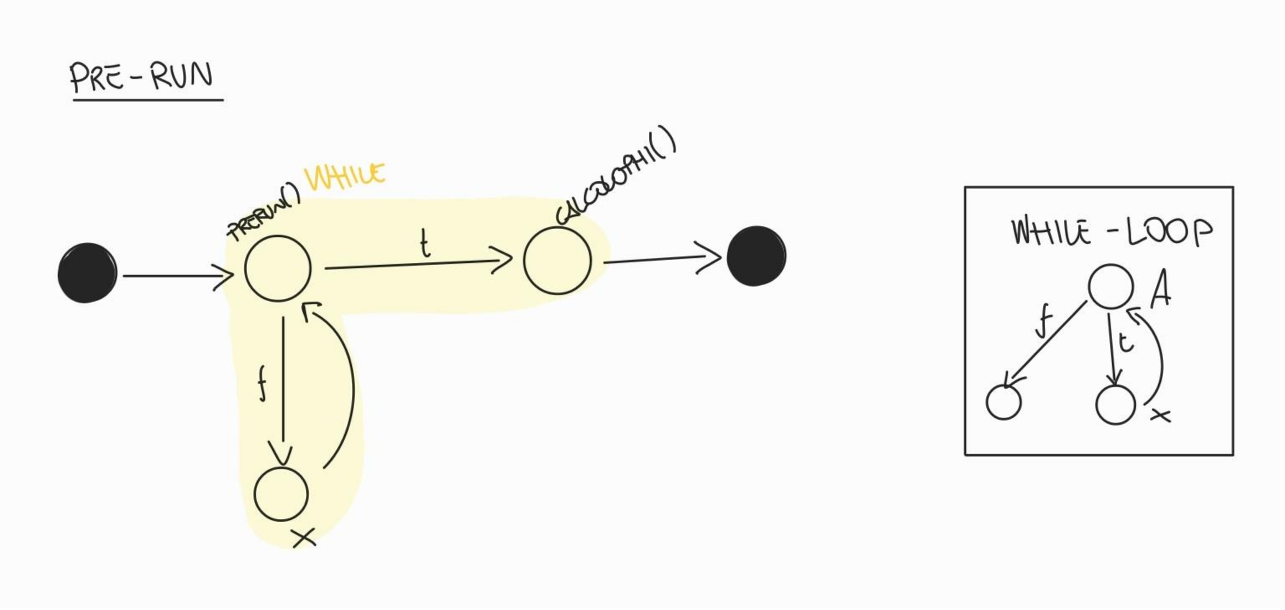
```
        }

        return lifetimes;
    }

    public double calcolophi0() {

      double totalFailureRate = n_Failure/totalTime; //calcolo del tasso 
complessivi di failure per difetto (failure/ora)
            double phi0 = totalFailureRate/N_0; //calcolo tasso iniziale 
di failure per difetto (failure /ora)

            return phi0;

    }

    public static void main(String[] args) {

        PreRun prerun = new PreRun();

        System.out.println("LifeTimes: " + 
Arrays.toString(prerun.prerun()) +"\nphi_0: "+ prerun.calcolophi0() + " 
failures/hour");
    }
}
```

Stampa esecuzione:

LifeTimes: [145.60837659245132, 207.78539181167912, 17.774155994904994, 2.43034491928478, 259.150480731169, 75.68389785698268, 81.83190082135859, 102.02289309203056, 266.0407620418409, 81.46044625095024, 73.79639282408536, 72.21829645923661, 70.57258301482779, 271.5967896124748, 168.44200575975862, 55.12596752884689, 121.00889648598296, 40.461143442734176, 10.815522495725766, 237.66288315550474, 288.667304483422, 75.2159439113121, 59.833830927227815, 6.683906442100325, 136.7858390296951, 300.89785950003716, 292.54330522016994, 454.4552186610682, 197.61170857142835, 71.76646582356783, 530.9722094073262, 149.57397974536383, 55.186499608127264, 87.20784371666252, 81.10382330196246, 101.50697522783295, 152.7383230621851, 332.5114523595682, 59.07488359121634, 50.86500500026752, 36.27125340481682, 7.04609758664137, 1.868652398993223, 194.45465606952038, 218.9314601687371, 1.0696773755047562, 92.87161428978943, 51.26520413183731, 24.909320408129208, 202.58950272173854, 121.18084077197054, 8.590549553721589, 215.31528334000095, 123.60674906799169, 148.30891192081708, 363.8521722057841, 377.7843611508584, 269.83666401840276, 394.48327816348944, 269.99856390770043, 12.620389513322428, 27.535747667652828, 100.30677282362839, 7.595172836720967, 74.91464090321725, 1.5878273720716793, 41.53569960996841, 117.60134891112727, 499.5652344553972, 25.454482082827447, 332.7606465290076, 62.18649492482161, 30.057398569255167, 216.2363137172397, 30.07836934939273, 217.56119587001749, 194.95001851821198, 143.97770168617993, 22.843608506481992, 149.21908593305534, 252.34498814591257, 30.72546538256614, 14.708241263312816, 91.93027365107108, 170.51454770197392, 16.205157195133292, 225.06237889048958, 23.31354837005494, 47.813852694077816, 162.58693757309257, 105.40677917642171, 229.58809700752042, 124.69721752185589, 508.1069343347759, 217.32166756201215, 12.432270454425923, 43.290228988012814, 142.78891498034972, 7.064313773144597, 85.28423395665203, 77.78509170011371, 368.6441086489752, 67.13321019488332, 147.5532673986047, 192.87596687612015, 10.944791867173427, 24.442290015841625, 95.17703110903737, 354.3179349209785, 48.03127171417987, 60.221633681864, 107.76446572718488, 10.18622230915538, 157.27024744979084, 319.71891344329885, 153.24161231552503, 65.11685409308734, 187.85829549303864, 189.59241722329548, 30.432524179178987, 262.0900120869107, 276.7574975622344, 117.75249552758603, 32.90342661942616, 497.66784477622684]

phi_0: 1.4757679421640926E-4 failures/hour

Flowgraph del codice:



Risposta 2
Simulare poi un **run** con T_i esponenziali di tasso λ_i (failure/ora) calcolato, ad ogni i, secondo J&M e di durata sufficiente a ottenere 50 failure, e rilevare la sequenza (T_1, …, T_{50}) dei tempi di lifetime.

Simuliamo un run generando casualmente n = 50 tempi di lifetime esponenziali di tasso λ_i (failure/ora). Si calcola λ_i ad ogni *i* secondo J&M con

$\lambda_i = 1/h_i$

Dove

$h_i = \phi(N - i + 1)$

Si ottengono i valori T_1, …, T_{50} che sarano poi inseriti nelle equazioni maximum likelihood per applicare Newton-Raphson.

Di seguito il codice java.
In breve, quello che è stato fatto nel codice, è stato costruire la classe Run che contenesse gli attributi necessari alla simulazione della fase di run con il modello J&M per generare 50 lifetime.

Tramite un main si va a richiamare il valore di φ_0 calcolato nel prerun che viene poi utilizzato nel modello J&M per calcolare λ_i ad ogni iterazione. Di conseguenza saranno generati i lifetime di tasso λ_i .
Saranno poi stampati a schermo i valori dei lifetime generati.

```
package Prit2_MQS_Pascale;

import java.util.Arrays;

public class Run {
    private static final int N_0 = 48;
    private static final int n_Failures = 50;
    private static final long seme = 57;

    public static void main(String[] args) {
        // Eseguire il pre-run per ottenere phi0
        PreRun preRun = new PreRun();
        preRun.prerun();
        double phi0 = preRun.calcolophi0(); // Calcola phi0

        // Simulazione del run con la formula di J&M
        int i = 1;
        double[] runLifetimes = new double[n_Failures];
        GeneratoreCasuale randomGenerator = preRun.RandomGenerator;

        while (i < n_Failures + 1) {
            double lambda_i = 1 / (phi0 * (N_0 - i + 1)); //formula J&M
            randomGenerator = new GeneratoreCasuale(lambda_i,
randomGenerator.getSuccSeme());
            double lifetime = randomGenerator.getNumberExp();
            runLifetimes[i-1] = lifetime;

            i++;
        }

        // Stampa dei risultati del run
        System.out.println("Lifetimes: " +
Arrays.toString(runLifetimes));
    }
}
```

Stampa esecuzione:

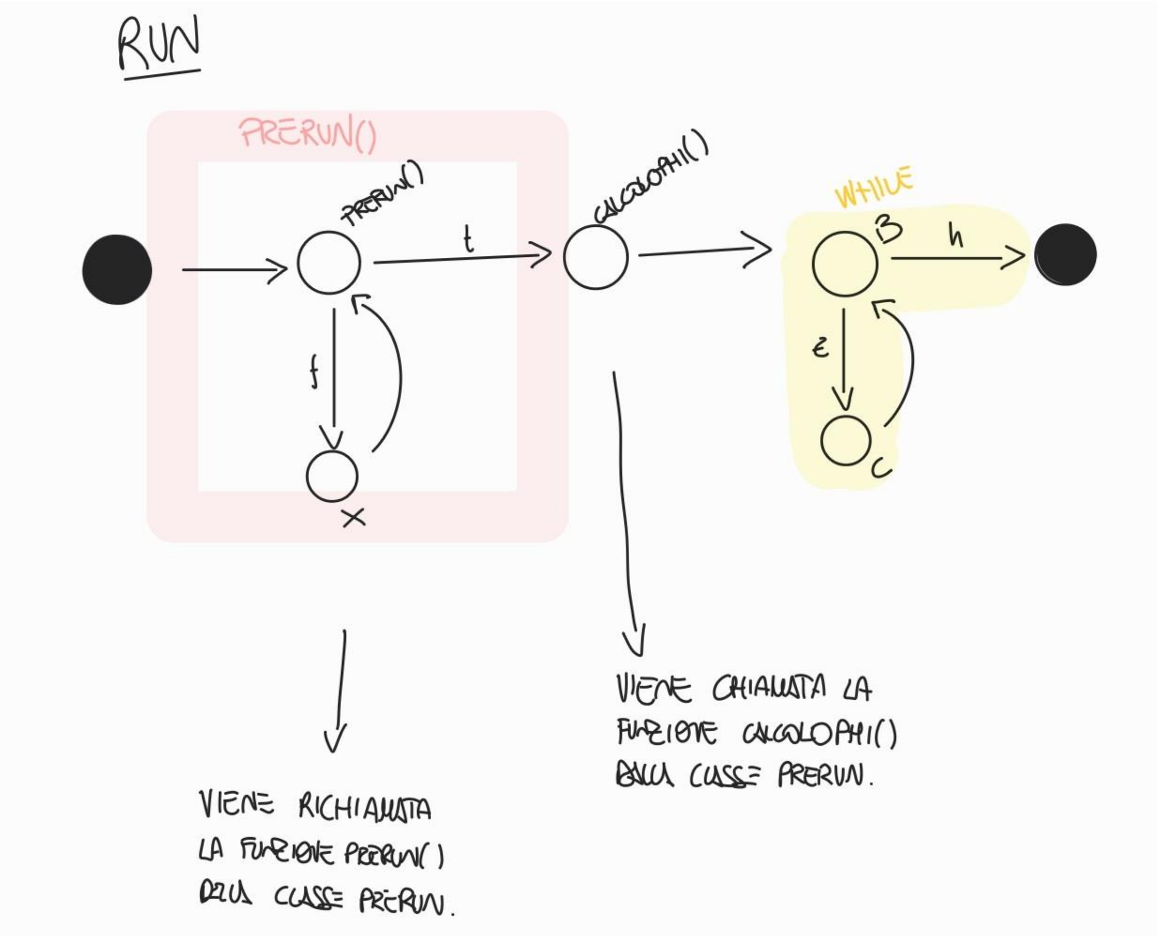
Lifetimes: [66.86425172035771, 10.519194530317042, 59.518190700394655, 157.11954434871183, 23.557984283434489, 170.82975503770652, 50.9726145848277, 307.9327167873723, 215.41804406493975, 69.9131301456767, 639.6678236957316, 30.74537950706204, 114.76346808843152, 171.31056954578605, 215.52639330456313, 84.41966979732078, 171.09830239589806, 644.5660662648901, 0.21861044615756725, 537.6042849011963, 1053.7112160601605, 117.58242518324928, 42.45226473485493, 630.6801656908518, 531.9455176338884, 385.23045644896456, 112.27590413645251, 33.26400585023092, 468.68908107500346, 172.80722355405035, 170.58617332661498, 305.3171898678347, 989.2687068342506, 44.686921844953375, 194.6371604042412, 714.373636223875, 400.29908963173307, 203.25165090077772, 62.36492510116009, 780.1637405277306, 53.21079376251557, 1722.5463042004783, 196.06385514847224, 416.2693594298822, 443.6465343747997, 1788.3163583762553, 333.11362669489955, 4035.2060474775994, Infinity, -3116.290249919203]

Nota sui risultati: alla fine della sequenza è possibile notare dei valori anomali: Infinity, -3116.290249919203. In J&M, abbiamo detto che si calcola λ_i ad ogni *i* secondo J&M con $\lambda_i = 1/h_i$.

Dove $h_i = \varphi(N - i + 1)$

Nel nostro caso specifico, abbiamo N = 48 e 50 failure da simulare, di conseguenza arriverà il momento in cui h_i assumerà valori negativi o pari a 0. Di conseguenza, λ_i avrà al denominatore 0 o dei valori negativi, facendo sì che nei LifeTime compaiano valori come Infinity e -3116.290249919203. Nei casi $h_{49} = \varphi(48 - 49 + 1)$ e $h_{50} = \varphi(48 - 50 + 1)$. I valori T_{49} e T_{50} saranno scartati.

Flowgraph del codice:



Risposta 3

Sulla base delle sequenze di LifeTime T_i e FailureTime t_i generate in fase di run, usando maximum likelihood e modello J&M, stimiamo i parametri N e φ. Sarà adottato il procedimento Newton-Raphson per la ricerca delle radici, e usati come valori inziali $N_0 = 48$ e $\varphi_0 = 0.00014757679421640926$, calcolato nel prerun.

```
In[36]:=

T = {66.86425172035771, 10.519194530317042, 59.518190700394655, 157.11954434871183,
  23.55798428343489, 170.82975503770652, 50.9726145848277, 307.9327167873723,
  215.41804406493975, 69.9131301456767, 639.6678236957316, 30.74537950706204,
  114.76346808843152, 171.31056954578605, 215.52639330456313, 84.41966979732078,
  171.09830239589806, 644.5660662648901, 0.21861044615756725, 537.6042849011963,
  1053.7112160601605, 117.58242518324928, 42.45226473485493, 630.6801656908518,
  531.9455176338884, 385.23045644896456, 112.27590413645251, 33.26400585023092,
  468.68908107500346, 172.80722355405035, 170.58617332661498, 305.3171898678347,
  989.2687068342506, 44.686921844953375, 194.6371604042412, 714.37363622387 5,
  400.29908963173307, 203.25165090077772, 62.36492510116009, 780.1637405277306,
  53.21079376251557, 1722.5463042004783, 196.06385514847224, 416.2693594298822,
  443.6465343747997, 1788.3163583762553, 333.11362669489955, 4035.2060474775994}


(* J&M *)
F[{c_, y_}] = { Sum[1 / (c - i + 1) - y * T[[i]], {i, 1, n}],
c / y - Sum[(c - i + 1) * T[[i]], {i, 1, n}]
};

jacobian[{c_, y_}] = Transpose[{D[ F[{c, y}], c] , D[ F[{c, y}], y]}];

NewtonSystem[X0_, max_] := Module[{}, n = Length[T];
  k = 0;
  Dp = {0, 0};
  P0 = X0;
  F0 = F[P0];
  Print["F[", P0, "]=", N[F0, 3]];
  P1 = P0;
  F1 = F0;

  While[k < max, k = k + 1;
   P0 = P1;
   F0 = F1;
   J0 = jacobian[P0];
   J0 = Rationalize[J0, 0];
   det = Det[J0];
   If[det == 0, Dp = {0, 0}, Dp = Inverse[J0].F0];
   P1 = P0 - Dp;
   Print["Dist. radici =", EuclideanDistance[P0, P1]];
```

```
    F1 = F[P1];
    Print["F[", P1, "]=", N[F1, 3]];
  ];
 ];

NewtonSystem[{48, 0.0001475767942164092 6}, 6]
```

Out[36]=

```
{66.8643, 10.5192, 59.5182, 157.12, 23.558, 170.83, 50.9726, 307.933,
 215.418, 69.9131, 639.668, 30.7454, 114.763, 171.311, 215.526, 84.4197,
 171.098, 644.566, 0.21861, 537.604, 1053.71, 117.582, 42.4523, 630.68,
 531.946, 385.23, 112.276, 33.264, 468.689, 172.807, 170.586, 305.317,
 989.269, 44.6869, 194.637, 714.374, 400.299, 203.252, 62.3649, 780.164,
 53.2108, 1722.55, 196.064, 416.269, 443.647, 1788.32, 333.114, 4035.21}

F[{48, 0.000147577}]={1.48593, 24 470.7}

Dist. radici =0.84032

F[{48.8403, 0.000153583}]={0.490619, 294.966}

Dist. radici =0.865541

F[{49.7059, 0.000148026}]={0.134829, 643.428}

Dist. radici =0.420236

F[{50.1261, 0.00014583}]={0.0154192, 117.826}

Dist. radici =0.0592869

F[{50.1854, 0.000145545}]={0.000242066, 2.10215}

Dist. radici =0.000951273

F[{50.1863, 0.000145541}]={6.05323 × 10^-8, 0.000520775}

Dist. radici =2.38176 × 10^-7

F[{50.1863, 0.000145541}]={3.10862 × 10^-15, 2.91038 × 10^-11}
```

Nella prima parte del codice vediamo inseriti i valori delle sequenze di LifeTime T_i generati. **Nota**: i valori negativi e infiniti sono stati rimossi.
Successivamente è presente il codice per applicare il metodo J&M per il calcolo di f1 ed f2. Di seguito il codice per l'applicazione del procedimento Newton-Raphson fornito dal testo.
La parte di codice: *NewtonSystem[{48, 0.0001475767942164092 6}, 6}* indica i valori iniziali N = 48, il valore di φ_0= 0.00014757679421640926 ottenuto nel prerun.

Il valore 6 indica il numero di iterazioni necessarie nel caso specifico in esame affinché la distanza tra le radici osservata sia inferiore a 10^{-6}, si sarebbero potute rendere necessarie ulteriori iterazioni in altri casi. Alla sesta iterazione la distanza tra le radici è pari a 2.38176x10^{-7}.

Per quanto riguarda la stampa dei **risultati** invece, i valori sono:

- N = 50.1863 arrotondato a 50
- φ = 0.000145541

Per quanto riguarda la F troviamo negli argomenti le radici N e φ ad ogni iterazione, infatti nella prima riga troviamo i valori iniziali inseriti nel codice. Nelle righe successive ci sono le radici

trovate dal procedimento. È possibile notare come il procedimento converga qualche iterazione prima che venga fermato.
A destra delle parentesi, nelle parentesi graffe, troviamo due valori, ovviai i valori assunti da f1 ed f2 con l'applicazione di Newton-Raphson. Nel caso di J&M:

- $f_1 = \sum_{i=1}^{1}(\frac{1}{N-i+1}) - \sum_{i=1}^{n}(\hat{\phi}T_i)$
- $f_2 = \frac{n}{\hat{\phi}} - \sum_{i=1}^{1}(\frac{1}{N-i+1})T_i$

Risposta 4

Innanzitutto:

- Il parametro N rappresenta una stima del numero effettivo di difetti presenti nel prodotto software.
- Il parametro φ rappresenta una stima del tasso effettivo di failure per difetto del prodotto (failure/ora).

Le funzioni f_1 e f_2 utilizzate per il procedimento Newton-Raphson sono state calcolate in questo modo:

si assume che: $P(E) = f(T_1)\, f(T_2) \dots f(T_n)$ (cioè che siano indipendenti), che nell'ipotesi di stima si scrive: $P(E|P) = f(T_1\,|P)\, f(T_2|P) \dots f(T_n\,|P)$.

Il metodo maximum likelihood parte da questa espressione per pervenire alla stima di P. Analogamente nel caso di due parametri *(N, φ)*.

Ora ricordiamo che nel Modello J&M la funzione di densità è:

$$f_i(t) = \phi(N-i+1)e^{-\phi(N-i+1)t}$$

Nella notazione del maximum likelihood la si può riscrivere nella forma

$$f(T_i) = \phi(N-i+1)e^{-\Phi(N-i+1)T_i}$$

Lo stimatore si scrive così:

$$\hat{f}(T_i|(N,\phi)) = \phi(N-i+1)e^{-\Phi(N-i+1)T_i}$$

evidenziandone i parametri. La densità della sequenza E:

$$\hat{p}(E|(N,\phi)) = \prod_{i=1}^{n} \phi(N-i+1)e^{-\Phi(N-i+1)T_i}$$

Questa espressione, secondo il metodo descritto, va derivata rispetto ai parametri e annullata, ottenendo:

$$\begin{cases} \dfrac{d}{dN}\sum_{i=1}^{n} \ln \hat{f}(T_i|(N,\phi)) = 0 \\ \dfrac{d}{d\phi}\sum_{i=1}^{n} \ln \hat{f}(T_i|(N,\phi)) = 0 \end{cases}$$

dopo varie operazioni algebriche diventa:

$$\begin{cases} \sum_{i=1}^{n} \dfrac{1}{N-i+1} - \sum_{i=1}^{n} \phi T_i = 0 \\ \dfrac{n}{\phi} - \sum_{i=1}^{n} (N-i+1)T_i = 0 \end{cases}$$

Questo sistema di due equazioni in due incognite va risolto per φ e N. Tuttavia il sistema è non lineare e pertanto fornisce *n* soluzioni di φ e N che possono essere ridotte a:

$$\begin{cases} \dfrac{d^2}{dN^2}\hat{p}(E|(N,\phi)) < 0 \\ \dfrac{d^2}{d\phi^2}\hat{p}(E|(N,\phi)) < 0 \end{cases}$$

Ricorrendo al metodo Newton-Raphson è possibile ottenere un'unica soluzione per N e φ.

Risposta 5

Partendo dai valori sopra stimati:

- N = 50.1863 arrotondato a 50
- φ = 0.000145541

J&M
Calcolo del numero *i* di failures che si dovranno sperimentare prima di eliminare il 90% dei fault e tempo t_i con J&M. In J&M c'è l'assunzione di **perfect debug**, cioè si ipotizza che tutti i difetti vengano corretti e che la correzione di un difetto non ne introduca altri. Quindi ci saranno tante

failures quanti difetti eliminati. Per questa ipotesi, se ti è l'istante di correzione del difetto i-esimo, si potrà scrivere:

$$M(ti) = i \qquad \text{dove } i = 0.9\ N = 0.9 * 50 = 45 \text{ failures}$$

Usando la notazione MTTF del modello J&M, si può scrivere

QUESITO 5

$\varphi = 0.000145541 \qquad N = 50 \qquad i = 45$

$$E[t_i] = \sum_{K=1}^{i} MTTF_K = \sum_{K=1}^{i} \frac{1}{\varphi(N-K+1)} \cong \frac{1}{\varphi}\left(\ln N - \ln(N-i)\right) =$$

$$\frac{1}{0.000145541} \cdot \left(\ln 50 - \ln(50-45)\right) =$$

$$6870.916 \cdot (3.912 - 1.609) =$$

$$6870.916 \cdot 2,303 = 15\,823 \text{ ORE}$$

Risposta 6

QUESITO 6.

$$R_i(t) = e^{-\lambda_i t} = e^{-\varphi(N-i+1)t} = e^{-\varphi(50-45+1)t} = e^{-0.000873\,t}$$

$$R_{45}(3) = e^{-0.000873\,t} = e^{-0.000873 \cdot 3} = e^{-0.002619} = 0.997$$

$1 - 0.997 = 0.003$ CIOÈ 0.3% DI PROBABILITÀ CHE SI VERIFICHINO FAILURE PRIMA DI 3 ORE

$$R_{45}(9) = e^{-0.000873\,t} = e^{-0.000873 \cdot 9} = e^{-0.007857} = 0.992$$

$1 - 0.992 = 0.008$ CIOÈ 0.8% DI PROBABILITÀ CHE SI VERIFICHINO FAILURE PRIMA DI 9 ORE

Concluso alle ore 17:04 del 5 Giugno 2024

Firma